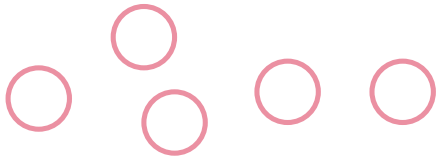
回顾：数据的逻辑结构

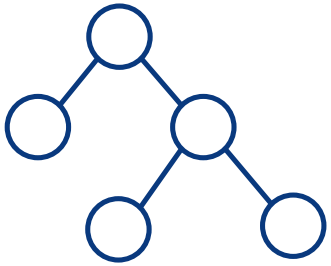
集合——数据元素间除“同属于一个集合”外，无其它

关系:

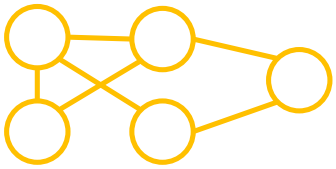


线性结构—— 一对一，如线性表、栈、队列

树形结构—— 一对多， 如树



图形结构—— 多对多，如图



第7章 图

7.1 图的定义和基本术语

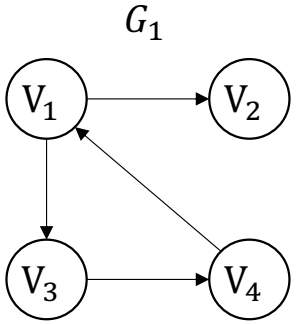
图: G=(V, VR)

V: 顶点(数据元素)的有穷非空集合;

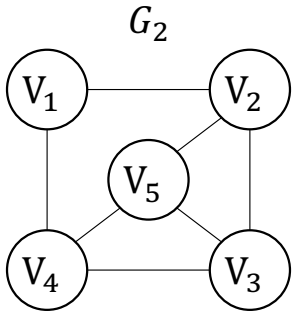
VR : 两个顶点之间的关系的有穷集合。

有向图: $< v, w > \in VR$ 表示从v到w的一条**弧**，且称v为弧尾(Tail)或初始点，称w为弧头(Head)或终端点，此时图为有向图。每条边都是有方向的。

无向图：若$< v, w > \in VR$必有$< w, v > \in VR$，即VR是**对称**的，则以无序对(v, w)代替这两个有序对，表示v和w之间的一条**边**，此时图为无向图。每条边都是无方向的。



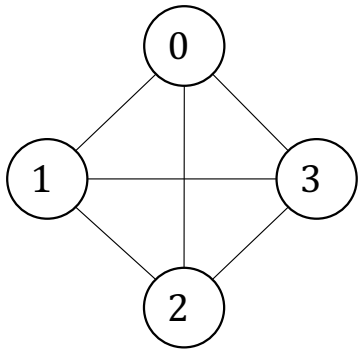
有向图



无向图

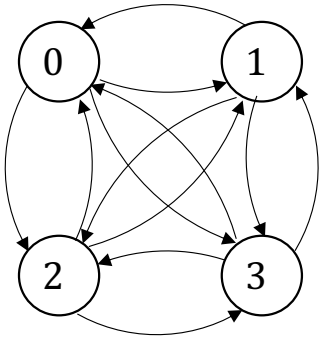
7.1 图的定义和基本术语

完全图: 任意两个点都有一条边相连



无向完全图

n个顶点，n(n-1)/2 条边



有向完全图

n个顶点， n(n-1) 条边

7.1 图的定义和基本术语

稀疏图: 有很少边或弧的图(e<nlogn) 。

稠密图: 有较多边或弧的图。

网: 边/弧带权的图。

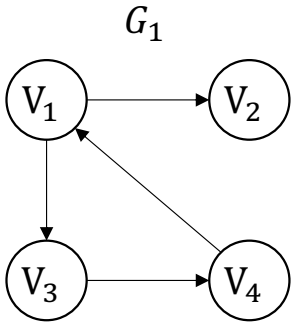
邻接: 有边/弧相连的两个顶点之间的关系。

存在(v_i, v_j)，则称v_i和v_j互为邻接点;

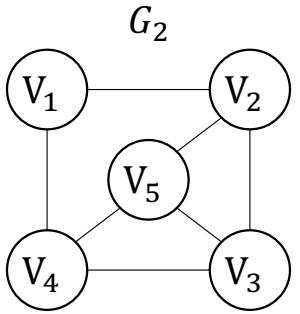
存在$< v_i, v_j >$，则称v_i邻接到v_j ， v_j邻接于v_i;

关联(依附): 边/弧与顶点之间的关系。

存在$(v_i, v_j)/< v_i, v_j >$，则称该边/弧关联于v_i和v_j;



有向图



无向图

7.1 图的定义和基本术语

顶点的度: 与该顶点相关联的边的数目，记为TD(v)

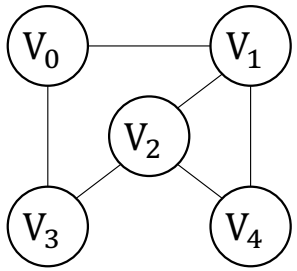
在有向图中，顶点的度等于该顶点的入度与出度之和。

$$e = \frac{1}{2}\sum_{i=1}^{n} TD(v_i)$$

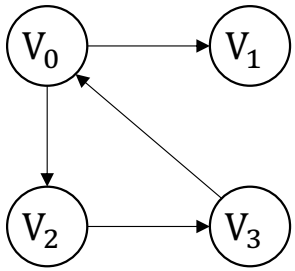
顶点v的入度是以v为终点的有向边的条数, 记作ID(v)

顶点v的出度是以v为始点的有向边的条数, 记作OD(v)

例: 图中各顶点的度，



顶点	度
V0	2
V1	3
V2	3
V3	2
V4	2

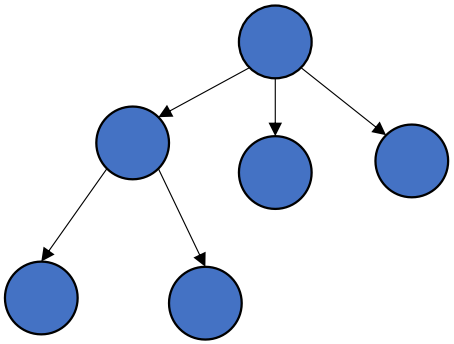


顶点	入度	出度	度
V0	1	2	3
V1	1	0	1
V2	1	1	2
V3	1	1	2

7.1 图的定义和基本术语

问: 当有向图中仅1个顶点的入度为0，其余顶点的入度均为1，此时是何形状?

答: 是树！而且是一棵有向树！



7.1 图的定义和基本术语

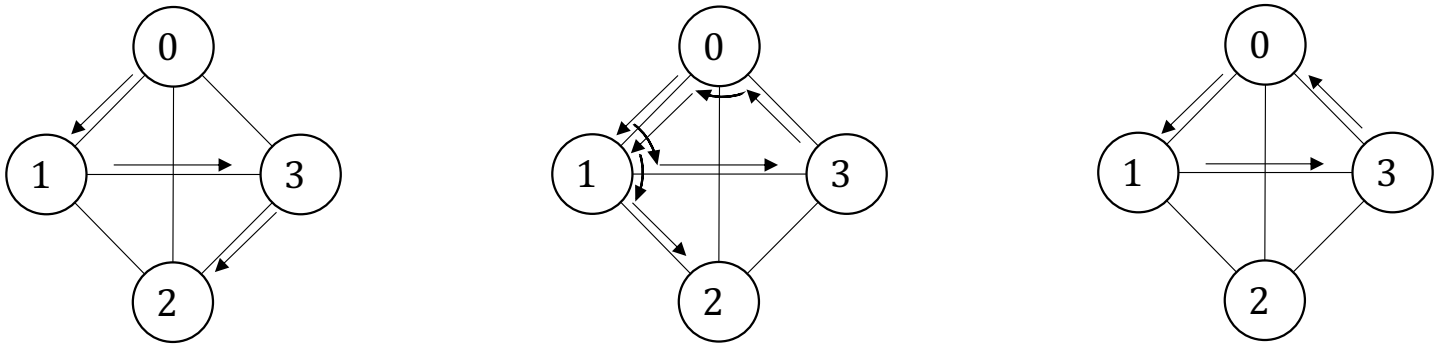
路径: 接续的边构成的顶点序列。

路径长度: 路径上边或弧的数目/权值之和。

回路(环): 第一个顶点和最后一个顶点相同的路径。

简单路径: 除路径起点和终点可以相同外，其余顶点均不相同的路径。

简单回路(简单环): 除路径起点和终点相同外，其余顶点均不相同的路径。



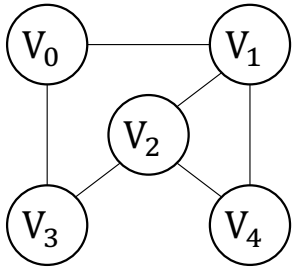
(a) 简单路径　　(b) 非简单路径　　(c) 回路

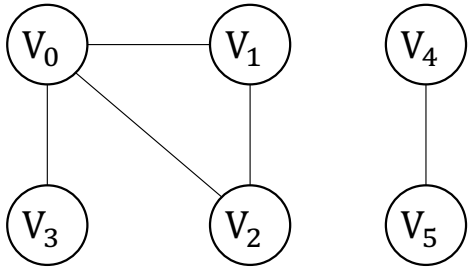
7.1 图的定义和基本术语

连通图(强连通图)

在无(有) 向图G=(V, {E})中，若对任何两个顶点v、u都存在从v到u的路径，则称G是连通图(强连通图)。

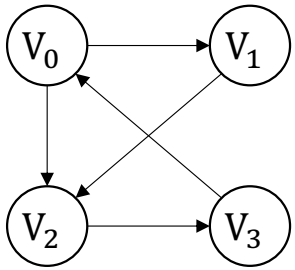
连通图

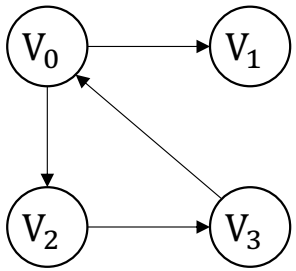




非连通图

强连通图





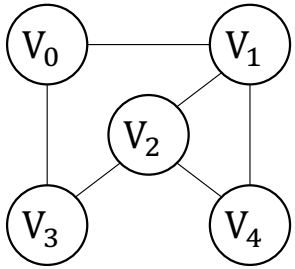
非强连通图

7.1 图的定义和基本术语

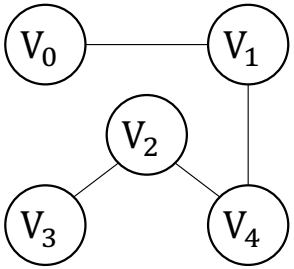
子图

设有两个图G= (V, {E}) 、G1= (V1, {E1}) , 若V1⊆V, E1 ⊆ E, 则称G1是G的子图。

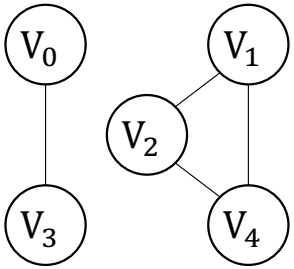
例: (b)、(c) 是(a)的子图。



(a)



(b)



(c)

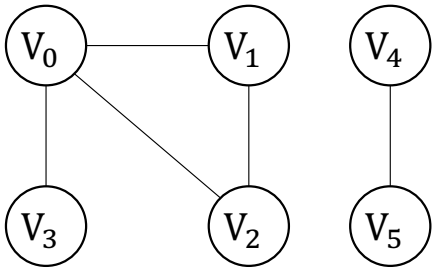
7.1 图的定义和基本术语

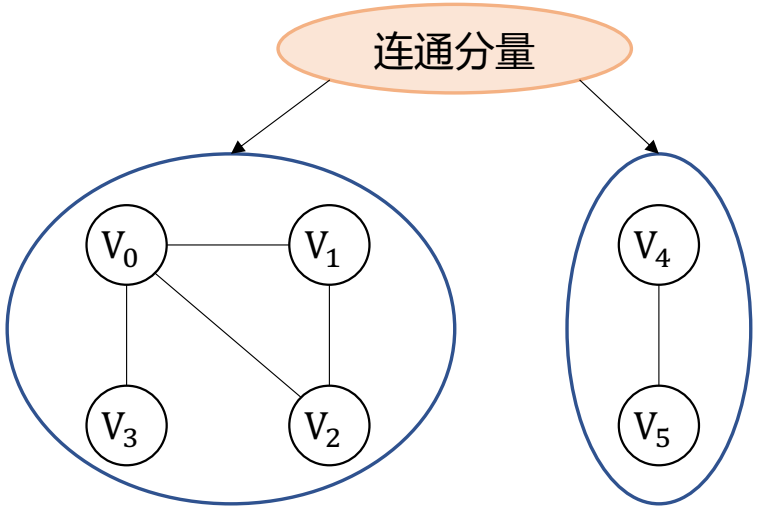
连通分量(强连通分量)

- 无向图G的极大连通子图称为G的连通分量。

极大连通子图意思是: 该子图是G连通子图，将G的任何不在该子图中的顶点加入，子图不再连通。

非连通图



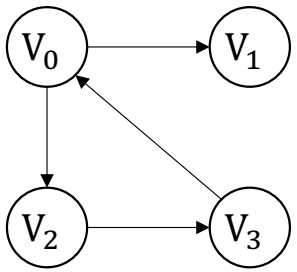


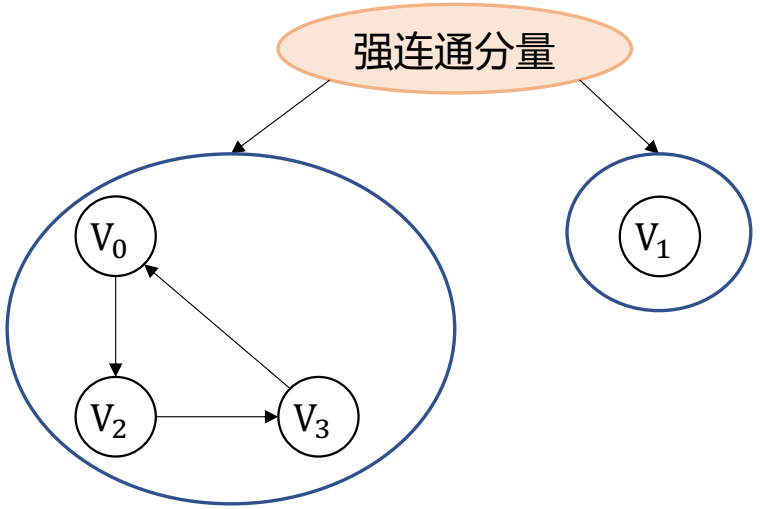
7.1 图的定义和基本术语

- 有向图G的极大强连通子图称为G的强连通分量。

极大强连通子图意思是:该子图是G的强连通子图，将G的任何不在该子图中的顶点加入，子图不再是强连通的。

非连通图



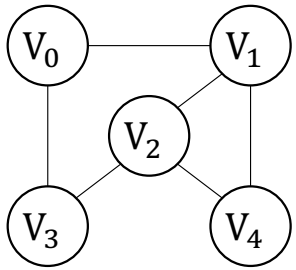


7.1 图的定义和基本术语

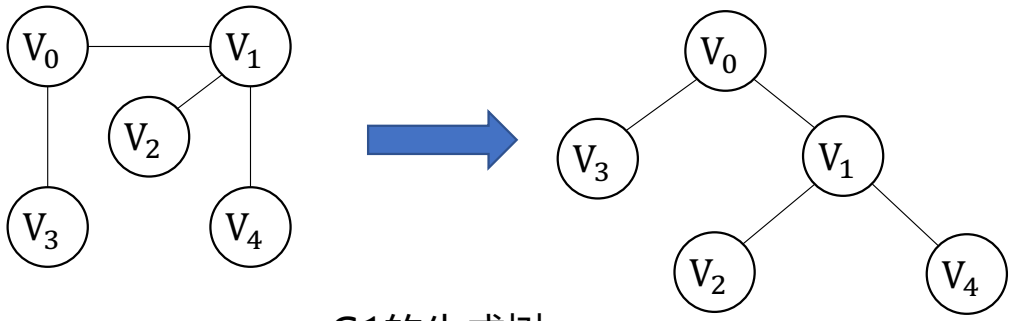
极小连通子图: 该子图是G的连通子图，在该子图中删除任何一条边，子图不再连通。

生成树: 包含无向图G所有顶点的极小连通子图。

生成森林: 对非连通图，由各个连通分量的生成树的集合。



连通图G1



G1的生成树

例：六度空间理论

“六度空间”理论又称作六度分隔(Six Degrees of Separation)理论。这个理论可以通俗地阐述为：“你和任何一个陌生人之间所间隔的人不会超过六个，也就是说，最多通过六个人你就能够认识任何一个陌生人。”该理论产生于20世纪60年代，由美国心理学家米尔格伦提出。

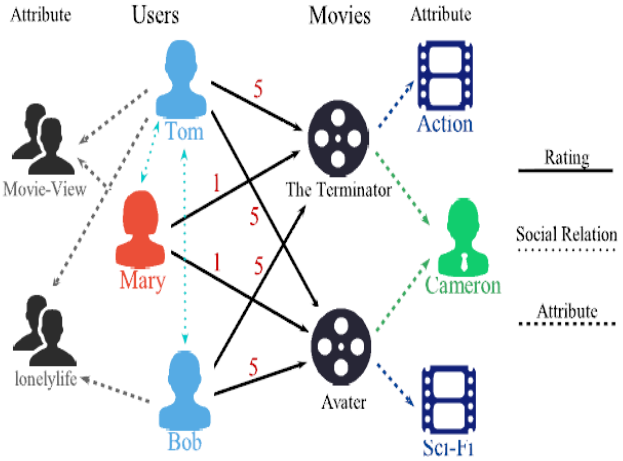
例：六度空间理论

但是米尔格兰姆的理论从来没有得到过严谨的证明，虽然屡屡应验，虽然很多社会学家一直都对其兴趣浓厚，但它只是一种假说。现在，许多科学家对此进行研究，它们都不约而同地使用了网络时代的新型通讯手段对“小世界现象”进行验证。

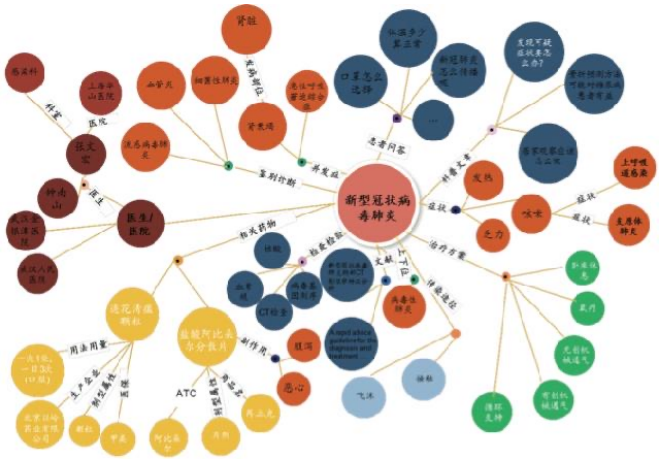
把六度空间理论中的人际关系网络抽象成一个无向图G。用图G中的一个顶点表示一个人，两个人认识与否用代表这两个人的顶点之间是否有一条边来表示。从任一顶点出发用广度优先方法对图进行遍历，统计所有路径长度不超过7的顶点

其他例子

社交网络



用户-电影异构信息网络



知识图谱

7.1 图的类型定义

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

图的抽象数据类型定义如下:

ADT Graph{

数据对象V：具有相同特性的数据元素的集合，称为顶点集。

数据关系R:: R={VR}

VR={<v,w>| v,w∈V 且 P(v,w),

<v,w>表示从v到w的弧，P(v,w)定义了弧<v,w>的信息

}

7.1 图的类型定义

基本操作P:

Create_Graph()：图的创建操作。

初始条件：无。

操作结果：生成一个没有顶点的空图G。

GetVex(G, v)：求图中的顶点v的值。

初始条件：图G存在，v是图中的一个顶点。

操作结果：生成一个没有顶点的空图G。

… …

7.1 图的类型定义

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

CreateGraph(&G,V,VR)

初始条件：V是图的顶点集，VR是图中弧的集合。

操作结果：按V和VR的定义构造图G。

DFSTraverse(G)

初始条件：图G存在。

操作结果：对图进行深度优先遍历。

BFSTraverse(G)

初始条件：图G存在。

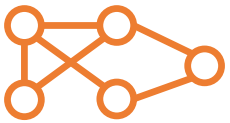
操作结果：对图进行广度优先遍历。

}ADT Graph

第7章 图

7.2 图的存储结构

图的逻辑结构：多对多



图没有顺序存储结构，
但可以借助二维数组来
表示元素间的关系

数组表示法（邻接矩阵）

链式存储结构： 多重链表

邻接表
邻接多重表
十字链表

重点介绍： 邻接矩阵（数组）表示法
邻接表（链式）表示法

7.2.1 邻接矩阵

1、数组（邻接矩阵）表示法

- 建立一个顶点表（记录各个顶点信息）和一个邻接矩阵（表示各个顶点之间关系）。
  - 设图A = （V, E）有n个顶点，则

顶点表Vexs[n]

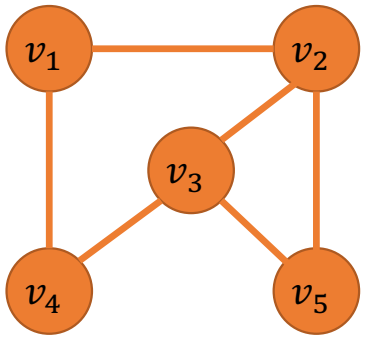
i	0	1	2	…	n-1
Vexs[i]	v_1	v_2	v_3	…	v_n

  - 图的邻接矩阵是一个二维数组A.arcs[n][n]，定义为：

$$A.arcs[i][j] = \begin{cases} 1, & \text{如果} < i, j > \in E \text{ 或者 } (i, j) \in E \\ 0, & \text{否则} \end{cases}$$

7.2.1 邻接矩阵

无向图的邻接矩阵表示法



邻接矩阵 A.arcs[i][j]=

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	1	0
v_2	1	0	1	0	1
v_3	0	1	0	1	1
v_4	1	0	1	0	0
v_5	0	1	1	0	0

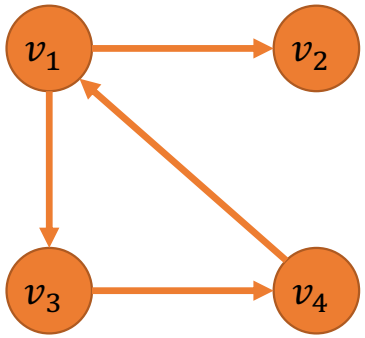
分析1：无向图的邻接矩阵是对称的；

分析2：顶点 i 的度=第i行（列）中1的个数；

特别：完全图的邻接矩阵中，对角元素为0，其余1。

7.2.1 邻接矩阵

有向图的邻接矩阵表示法



邻接矩阵

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

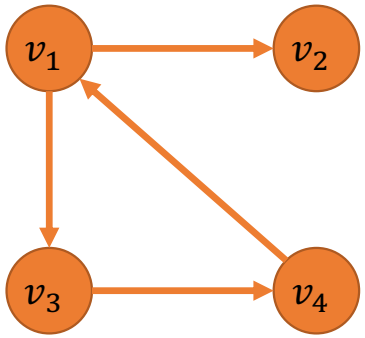
注：在有向图的邻接矩阵中，

第i行含义：以结点v_i为尾的弧（即出度边）；

第i列含义：以结点v_i为头的弧（即入度边）。

7.2.1 邻接矩阵

有向图的邻接矩阵表示法



邻接矩阵

	v_1	v_2	v_3	v_4
v_1	0	1	1	0
v_2	0	0	0	0
v_3	0	0	0	1
v_4	1	0	0	0

分析1：有向图的邻接矩阵是可能是不对称的；

分析2：顶点的出度=第i行元素之和

顶点的入度=第i列元素之和

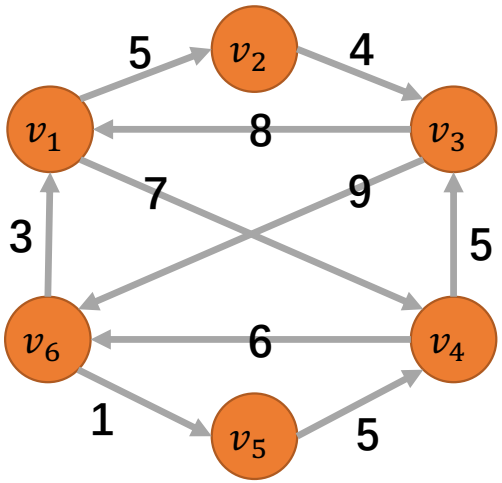
顶点的度=第i行元素之和 + 第i列元素之和

7.2.1 邻接矩阵

网（即有权图）的邻接矩阵表示法

定义为：

$$A.arcs[i][j]=\begin{cases} W_{ij} & <vi, vj> \text{ 或 } (vi, vj) \in VR \\ \infty & \text{无边（弧）} \end{cases}$$



邻接矩阵

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	∞	5	∞	7	∞	∞
v_2	∞	∞	4	∞	∞	∞
v_3	8	∞	∞	∞	∞	9
v_4	∞	∞	5	∞	∞	6
v_5	∞	∞	∞	5	∞	∞
v_6	3	∞	∞	∞	1	∞

7.2.1 邻接矩阵

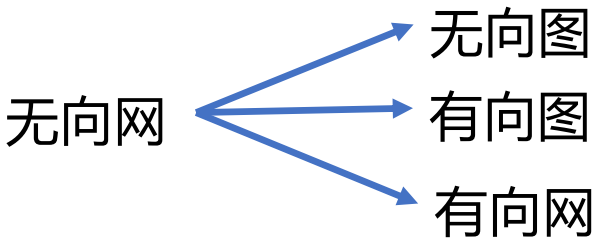
邻接矩阵的存储表示：用两个数组分别存储顶点表和邻接矩阵

```
#define MaxInt 32767                    //表示极大值，即∞
#define MVNum 100                       //最大顶点数
typedef char VerTexType;                //设顶点的数据类型为字符型
typedef int ArcType;                    //假设边的权值类型为整型

typedef struct{
    VerTexType vexs[MVNum];             //顶点表
    ArcType arcs[MVNum][MVNum];         //邻接矩阵
    int vexnum, arcnum;                 //图的当前顶点数和边数
}AMGraph;  //Adjacency Matrix Graph
```

7.2.1 邻接矩阵

2、采用邻接矩阵表示法创建无向网



【算法思想】

（1）输入总顶点数和总边数。

（2）依次输入顶点的信息存入顶点表中。

（3）初始化邻接矩阵，使每个权值初始化为极大值。

（4）构造邻接矩阵。

7.2.1 邻接矩阵

算法6.1 采用邻接矩阵表示法创建无向网

```
Status CreateUDN(AMGraph &G){    //采用邻接矩阵表示法，创建无向图G
    scanf (&G.vexnum, &G.arcnum);    //输入总顶点数，总边数
    for(i=0; i<G.vexnum; ++i)
        scanf (&G.vexs[i]);                //依次输入点的信息
    for(i=0; i<G.vexnum; ++i)        //初始化邻接矩阵
        for(j=0; j<G.vexnum; ++j)
            G.arcs[i][j] = MaxInt;       //边的权值均置为极大值
```

7.2.1 邻接矩阵

算法6.1 采用邻接矩阵表示法创建无向网

```
(续上页)
    for(k=0; k<G.arcnum; ++k){     //构造邻接矩阵
      scanf (&v1, &v2, &w);    //输入一条边所依附的顶点及边的权值
      i = LocateVex(G, v1);
      j = LocateVex(G, v2);  //确定v1和v2在G中的位置
      G.arcs[i][j] = w;          //边<v1, v2>的权值置为w
      G.arcs[j][i] = G.arcs[i][j];   //置<v1, v2>的对称边<v2, v1>的权值为w
    }//for
}//CreateUDN
```

7.2.1 邻接矩阵

1、数组（邻接矩阵）表示法

- 建立一个顶点表（记录各个顶点信息）和一个邻接矩阵（表示各个顶点之间关系）。
  - 设图A = （V, E）有n个顶点，则

顶点表Vexs[n]

i	0	1	2	…	n-1
Vexs[i]	v_1	v_2	v_3	…	v_n

  - 图的邻接矩阵是一个二维数组A.arcs[n][n]，定义为：

$$A.arcs[i][j] = \begin{cases} 1, & \text{如果} < i, j > \in E \text{ 或者 } (i, j) \in E \\ 0, & \text{否则} \end{cases}$$

7.2.1 邻接矩阵

算法6.1 采用邻接矩阵表示法创建无向网

```
(续上页)
    for(k=0; k<G.arcnum; ++k){     //构造邻接矩阵
      scanf (&v1, &v2, &w);    //输入一条边所依附的顶点及边的权值
      i = LocateVex(G, v1);
      j = LocateVex(G, v2);  //确定v1和v2在G中的位置
      G.arcs[i][j] = w;          //边<v1, v2>的权值置为w
      G.arcs[j][i] = G.arcs[i][j];   //置<v1, v2>的对称边<v2, v1>的权值为w
    }//for
}//CreateUDN
```

7.2.1 采用邻接矩阵表示法创建无向图

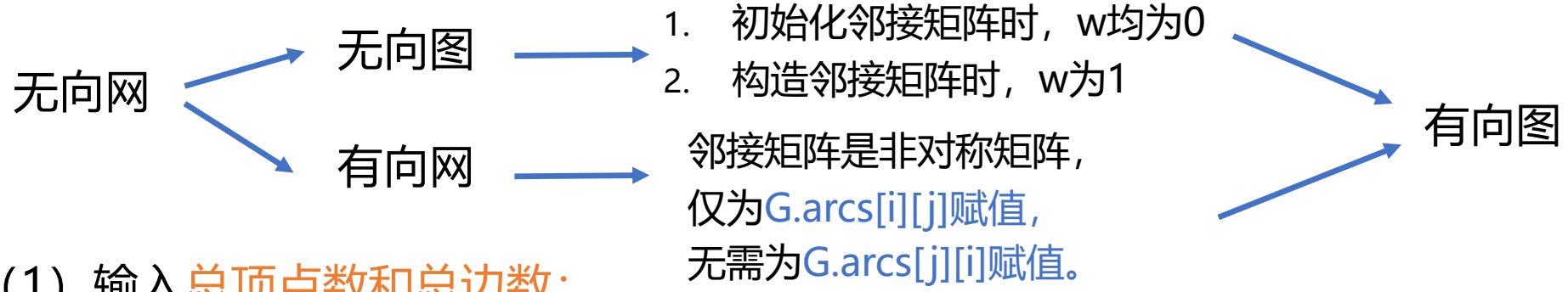
杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

补充算法：在图中查找顶点

```
int LocateVex(AMGraph G, VertexType u){
//图G中查找顶点u，存在则返回顶点表中的下标；否则返回-1
    int i;
    for(i=0; i<G.vexnum; ++i)
        if(u==G.vexs[i])    retrun i;
    return -1;
}
```

7.2.1 邻接矩阵

2、采用邻接矩阵表示法创建无向网



（1）输入总顶点数和总边数；

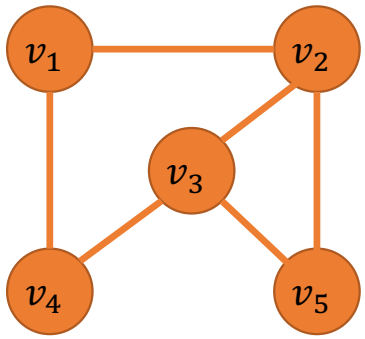
（2）依次输入顶点的信息存入顶点表中；

（3）初始化邻接矩阵，使每个权重初始化为极大值；

（4）构造邻接矩阵。

7.2.1 邻接矩阵

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$



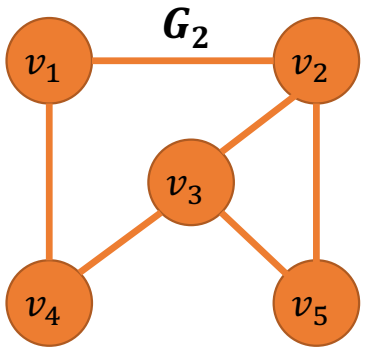
- 邻接矩阵——有什么好处?
  - 直观、简单、好理解
  - 方便检查任意一对顶点间是否存在边
  - 方便找任一顶点的所有“邻接点”（有边直接相连的顶点）
  - 方便计算任一顶点的“度”（从该点发出的边数为“出度”，指向该点的边数为“入度”）
    - 无向图：对应行（或列）非0元素的个数
    - 有向图：对应行非0元素的个数是“出度”；
      对应列非0元素的个数是“入度”。

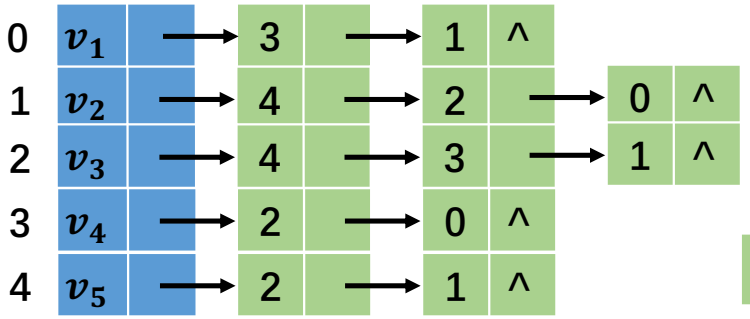
7.2.1 邻接矩阵

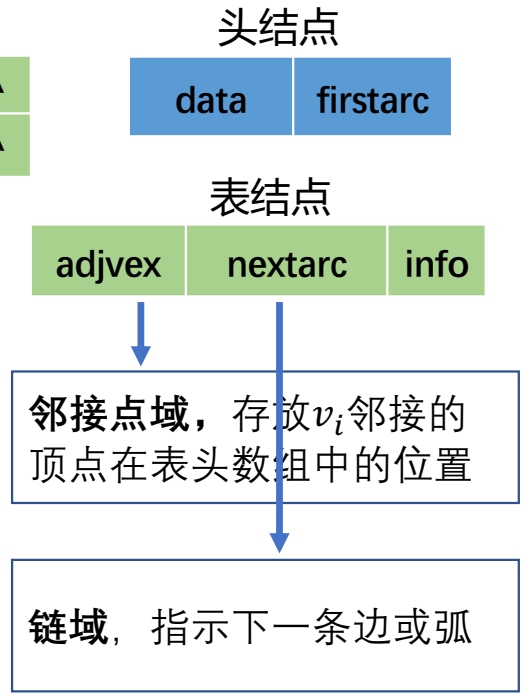
- 邻接矩阵——有什么不好?
  - 不便于增加和删除顶点
  - 浪费空间——存稀疏图（点很多而边很少）有大量无效元素
    - 对稠密图（特别是完全图）还是很合算的
  - 浪费时间——统计稀疏图中一共有多少边

7.2.2 邻接表

1、邻接表表示法（链式）



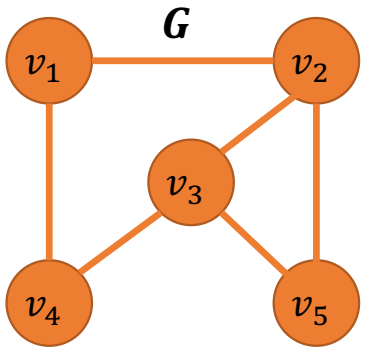


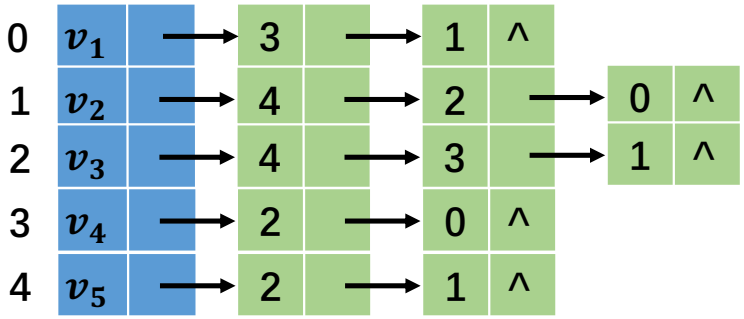


- 顶点：
  - 按编号顺序将顶点数据存储在一维数组中；
- 关联同一顶点的边（以顶点为尾的弧）：
  - 用线性链表存储

7.2.2 邻接表

无向图



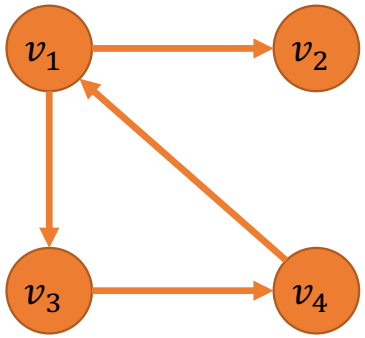


特点：

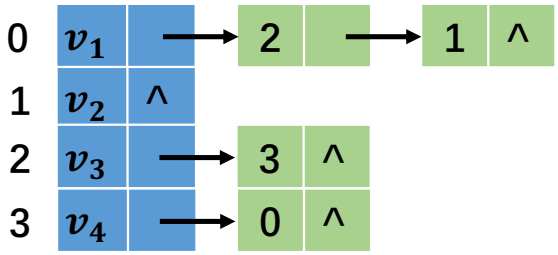
- 邻接表不唯一；
- 若无向图中有n个顶点、e条边，则其邻接表需n个头结点和2e个表结点，适合存储稀疏图。 O(n+2e)
- 无向图中顶点v_i的度为第i个单链表中的结点数

7.2.2 邻接表

有向图

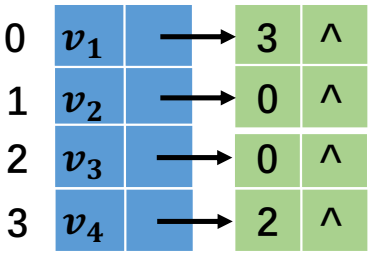


O(n+e) 邻接表



找出度易，找入度难

逆邻接表



找入度易，找出度难

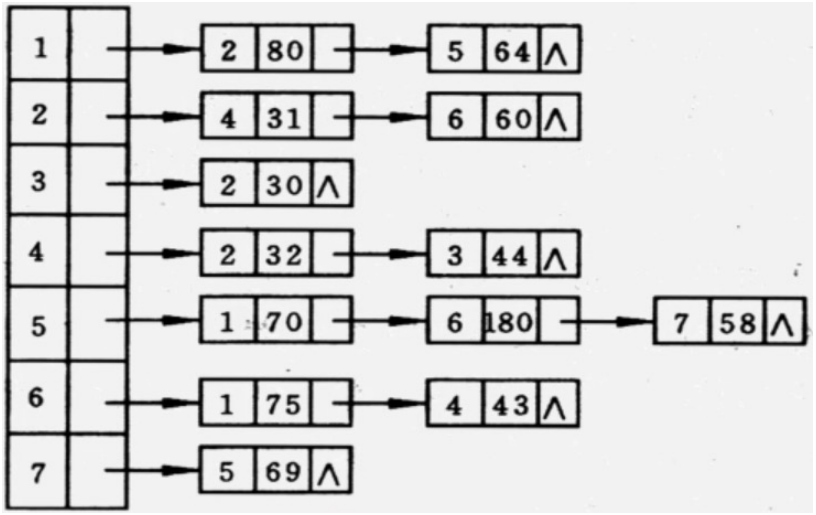
特点：

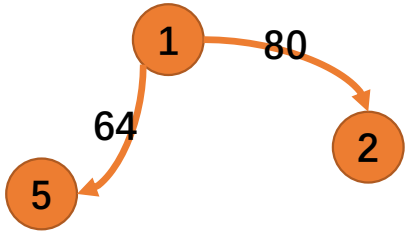
- 顶点v_i的**出度**为第i个单链表中的结点个数；
- 顶点v_i的**入度**为整个单链表中邻接点域值是i-1的结点个数。

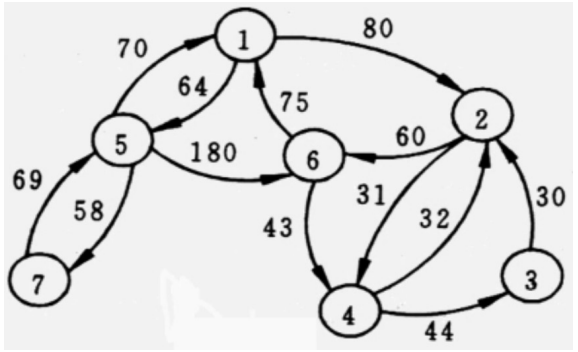
- 顶点v_i的**入度**为第i个单链表中的结点个数；
- 顶点v_i的**出度**为整个单链表中邻接点域值是i-1的结点个数。

练习

已知某网的邻接（出边）表，请画出该网络







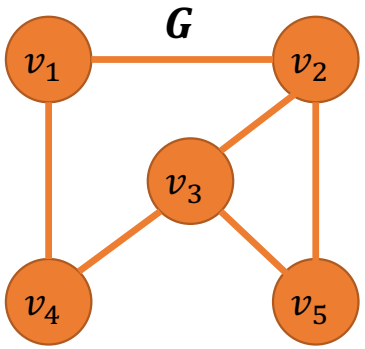
当邻接表的存储结构形成后，图便唯一确定

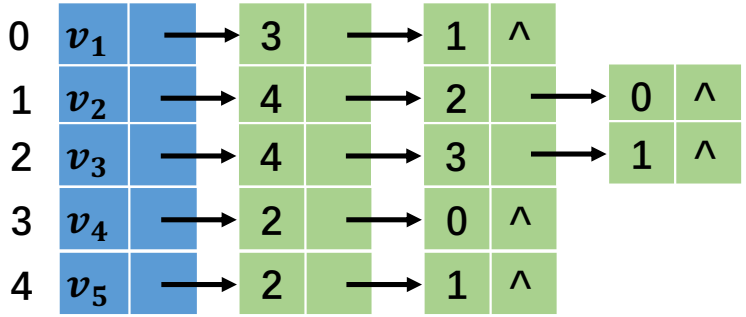
7.2.2 邻接表

图的邻接表存储表示：

顶点的结点结构

data	firstarc






```
typedef struct VNode{
    VerTexType  data;          // 顶点信息
    ArcNode     *firstarc;     // 指向第一条依附该顶点的边的指针
} VNode, AdjList[MVNum];       // AdjList表示邻接表类型
```

说明：例如，AdjList v;　相当于：VNode v[MVNum];

7.2.2 邻接表

图的邻接表存储表示：

弧（边）的结点结构

adjvex	nextarc	info

```
#define MVNum 100              //最大顶点数
typedef struct ArcNode{        //边结点
        int adjvex;            //该边所指向的顶点的位置
        struct ArcNode *nextarc;          //指向下一条边的指针
        OtherInfo info;                   //和边相关的信息
}ArcNode;
```

7.2.2 邻接表

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

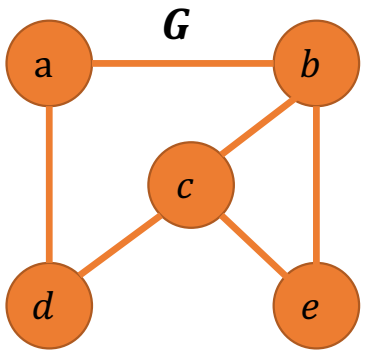
图的邻接表存储表示：

图的结构定义

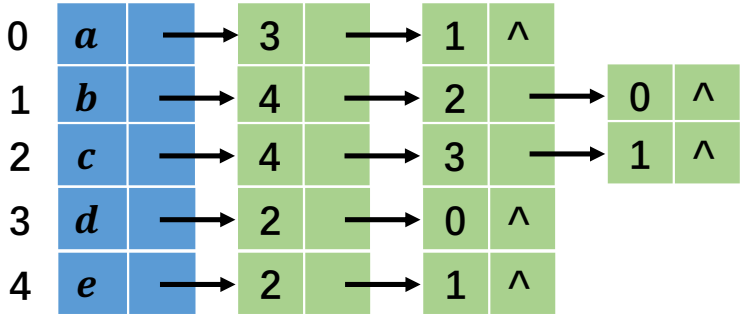
```
typedef struct ArcNode{
        AdjList     vertices;          //vertices：vertex的复数
        int vexnum, arcnum;            //图的当前顶点数和弧数
}ALGraph;
```

7.2.2 邻接表

邻接表操作举例说明：



邻接表




```
ALGraph G;              //定义了邻接表表示的图G
G.vexum=5; G.arcnum=6;  //图G包含5个顶点，6条边
G.vertices[1].data= 'b' ;  //图G中第2个顶点是b
p=G.vertices[1].firtarc;   //指针p指向边顶点b的第一条边结点
p->adjvex=4;               //p指针所指边结点是到下标为4的结点的边
```

7.2.2 邻接表

2、采用邻接表表示创建无向图

【算法思想】

（1）输入总顶点数和总边数

（2）建立顶点表

依次输入点的信息存入顶点表中

使每个表头结点的指针域初始化为NULL

（3）创建邻接表

依次输入每条边依附的两个顶点

确定两个顶点的序号i和j，建立边结点

将此边结点分别插入到v_i和v_j对应的两个边链表的头部

7.2.2 邻接表

算法6.2 采用邻接表表示法创建无向图

```
Status CreateUDG(ALGraph &G){  //采用邻接表表示法，创建无向图G
    scanf (&G.vexnum, &G.arcnum);  //输入总顶点数，总边数
    for(i=0; i<G.vexnum; ++i){      //输入各点，构造表头结点表
        scanf (&G. vertices[i].data);    //输入顶点值
        G. vertices[i].firstarc=NULL;  //初始化表头结点的指针域
    } //for
    for(k=0; k<G.arcnum; ++k){          //输入各边，构造邻接表
        scanf (&v1, &v2);                //输入一条边依附的两个顶点
        i = LocateVex(G, v1);
        j = LocateVex(G, v2);
```

(接下页)

7.2.2 邻接表

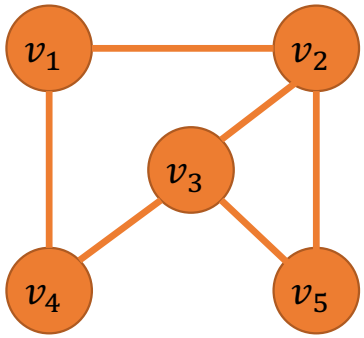
算法6.2 采用邻接表表示法创建无向图

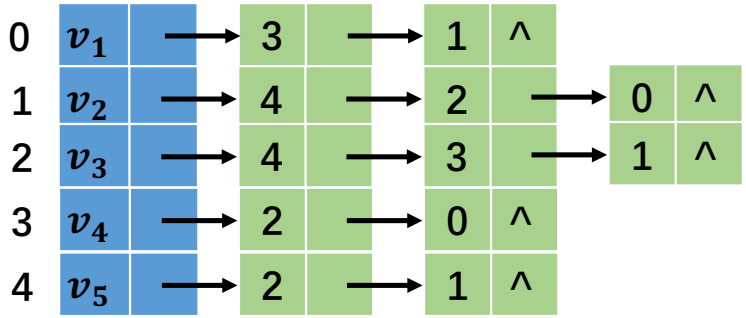
(接上页)

```
        p1=new ArcNode;          //生成一个新的边结点*p1
        p1->adjvex=j;            //邻接点序号为j
        p1->nextarc=G.vertices[i].firstarc;
        G.vertices[i].firstarc=p1;     //将新结点*p1插入顶点Vi的边表头部
        p2=new ArcNode;          //生成另一个对称的新的边结点*p2
        p2->adjvex=i;            //邻接点序号为i
        p2->nextarc=G.vertices[j].firstarc;
        G.vertices[j].firstarc=p2;     //将新结点*p2插入顶点Vj的边表头部
        } //for
        return OK;
}//CreateUDG
```

7.2.2 邻接表

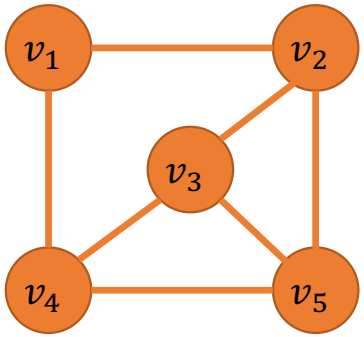
邻接表特点



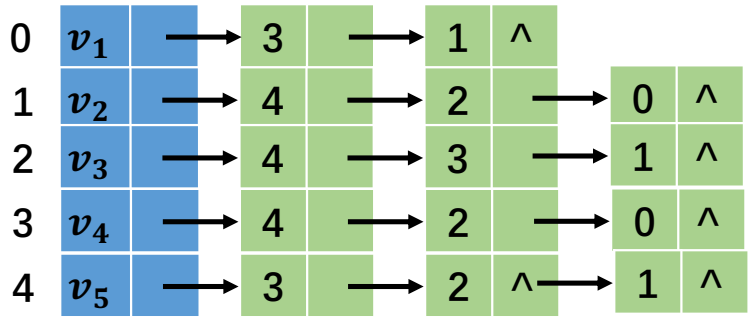


- 方便找任一顶点的所有“邻接点”
- 节约稀疏图的空间
  - 需要N个头指针+2E个结点（每个结点至少2个域）
- 方便计算任一顶点的“度”？
  - 对无向图：是的
  - 对有向图：只能计算“出度”；需要构造“逆邻接表”（存指向自己的边）来计算“入度”
- 不方便检查任意一对顶点间是否存在边

邻接矩阵与邻接表表示法的关系

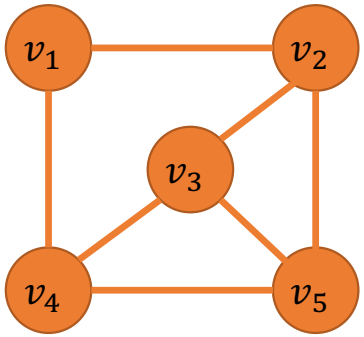


$$
\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\
\begin{pmatrix}
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0
\end{pmatrix}
\end{array}
$$

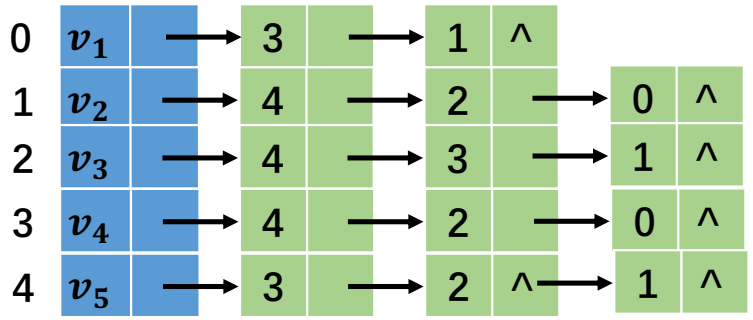


1、联系：邻接表中每个链表对应于邻接矩阵中的一行，链表中结点个数等于一行中非零元素的个数。

邻接矩阵与邻接表表示法的关系



	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	1	0
v_2	1	0	1	0	1
v_3	0	1	0	1	1
v_4	1	0	1	0	1
v_5	0	1	1	1	0



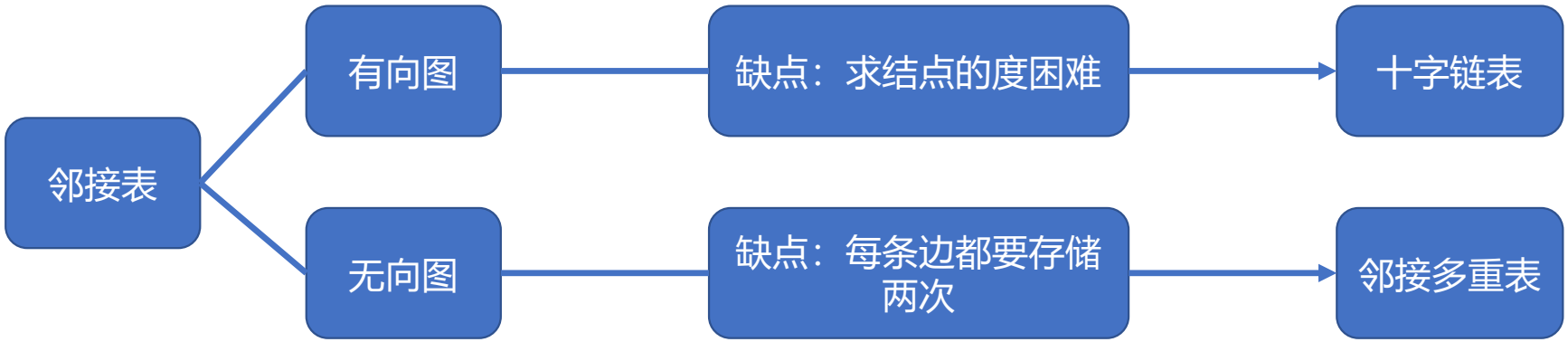
2、区别：

① 对于任一确定的无向图，邻接矩阵都是唯一的（行列号与顶点编号一致），但邻接表不唯一（连接次序与顶点编号无关）。

② 邻接矩阵的空间复杂度为$O(n^2)$，而邻接表的空间复杂度为$O(n+e)$。

3、用途：邻接矩阵多用于稠密图；而邻接表多用于稀疏图

图的存储结构



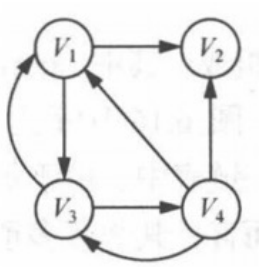
7.2.3 十字链表——用于有向图

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

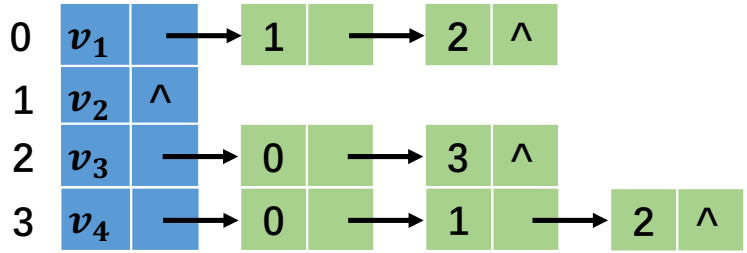
十字链表（Orthogonal List）是**有向图**的另一种链式存储结构。我们也可以把它看成是将有向图的邻接表和逆邻接表结合起来形成的一种链表。

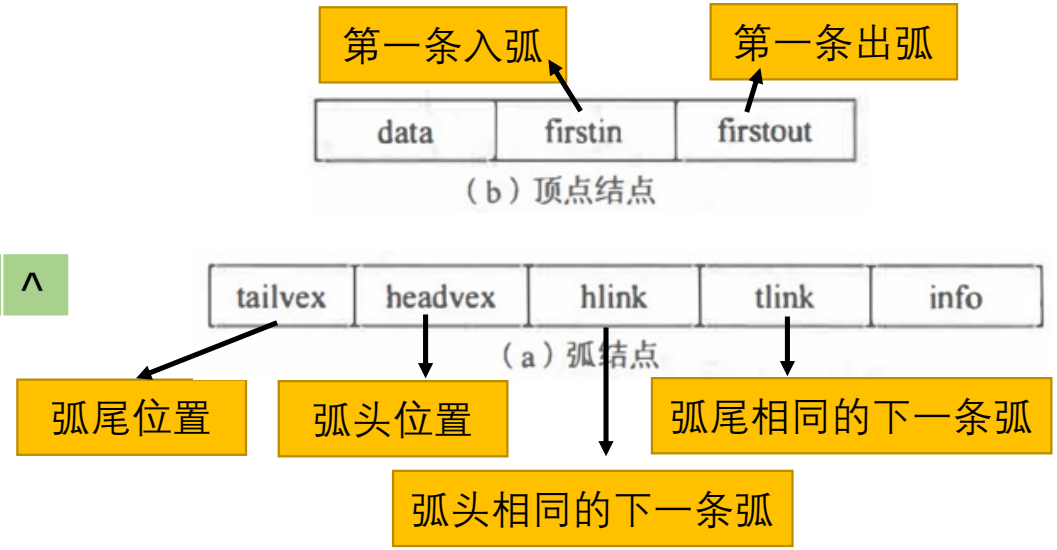
有向图中的每一条弧对应十字链表中的一个弧结点，同时有向图中的每个顶点在十字链表中对应一个结点，叫做顶点结点。

7.2.3 十字链表

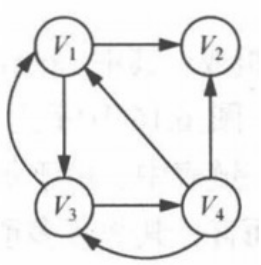

V_1
V_2
V_3
V_4

邻接表

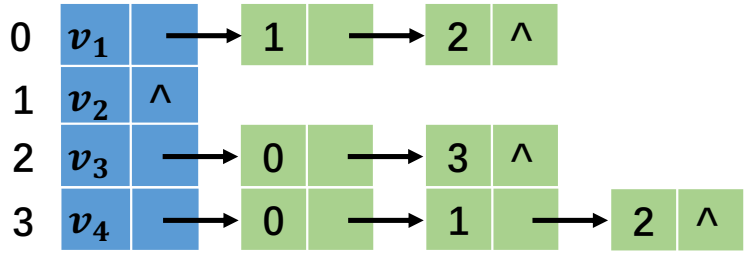

0
v_1
1
2
^
1
v_2
^
2
v_3
0
3
^
3
v_4
0
1
2
^

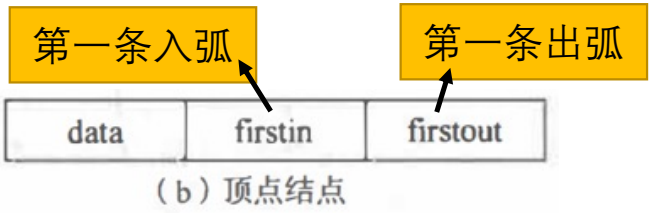

第一条入弧
第一条出弧
data
firstin
firstout
（b）顶点结点
tailvex
headvex
hlink
tlink
info
（a）弧结点
弧尾位置
弧头位置
弧尾相同的下一条弧
弧头相同的下一条弧

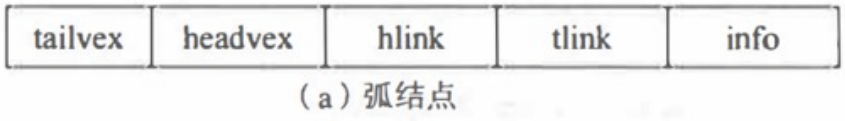
7.2.3 十字链表

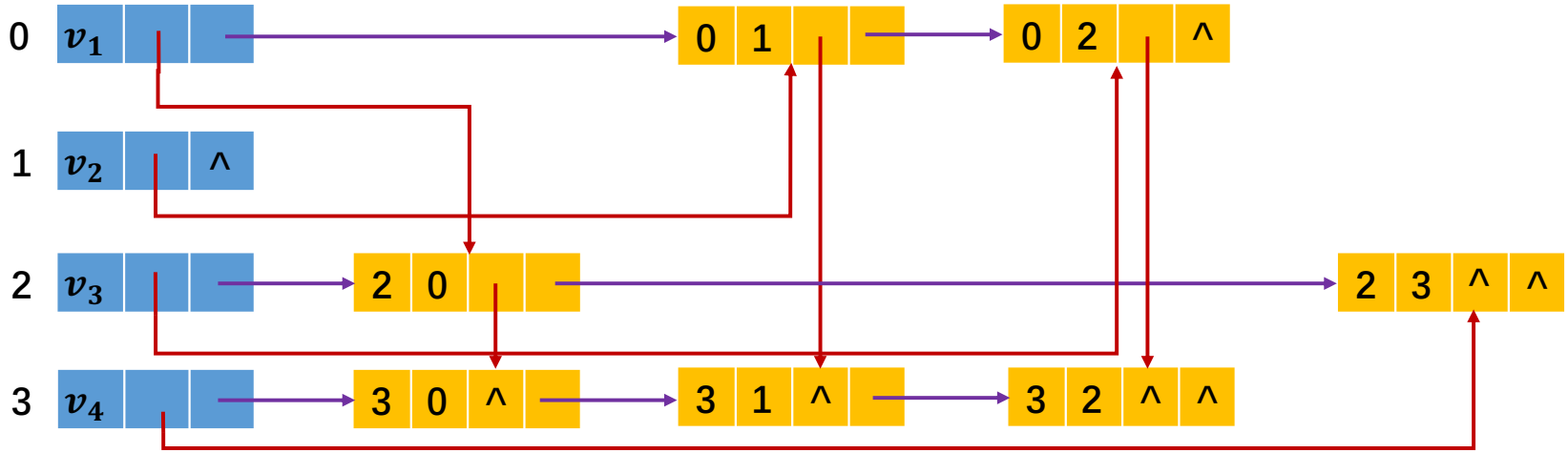

V_1
V_2
V_3
V_4

邻接表

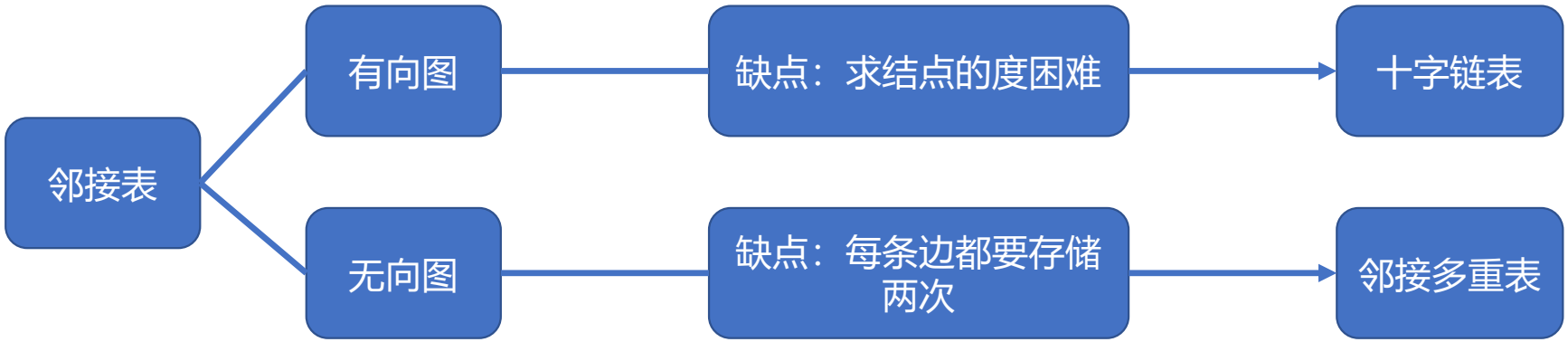

0 v_1 1 2 ^
1 v_2 ^
2 v_3 0 3 ^
3 v_4 0 1 2 ^


第一条入弧
第一条出弧
data
firstin
firstout
（b）顶点结点


tailvex
headvex
hlink
tlink
info
（a）弧结点


0 v_1 0 1 0 2 ^
1 v_2 ^
2 v_3 2 0 2 3 ^ ^
3 v_4 3 0 ^ 3 1 ^ 3 2 ^ ^

图的存储结构



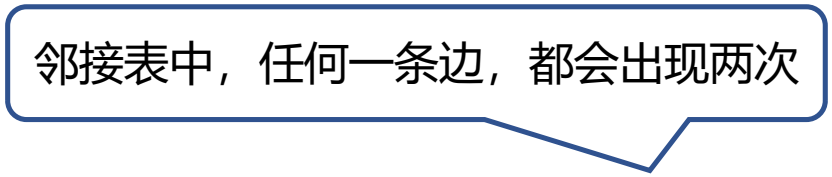
7.2.4 邻接多重表(无向图链式存储结构)

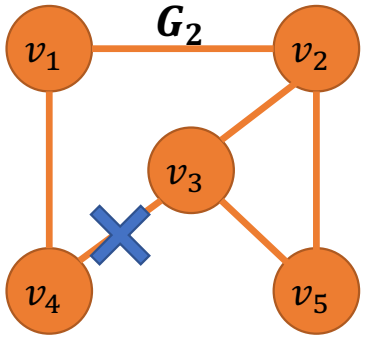
杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

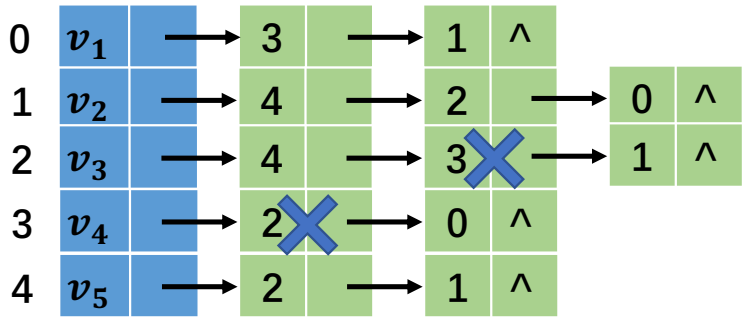
回顾：

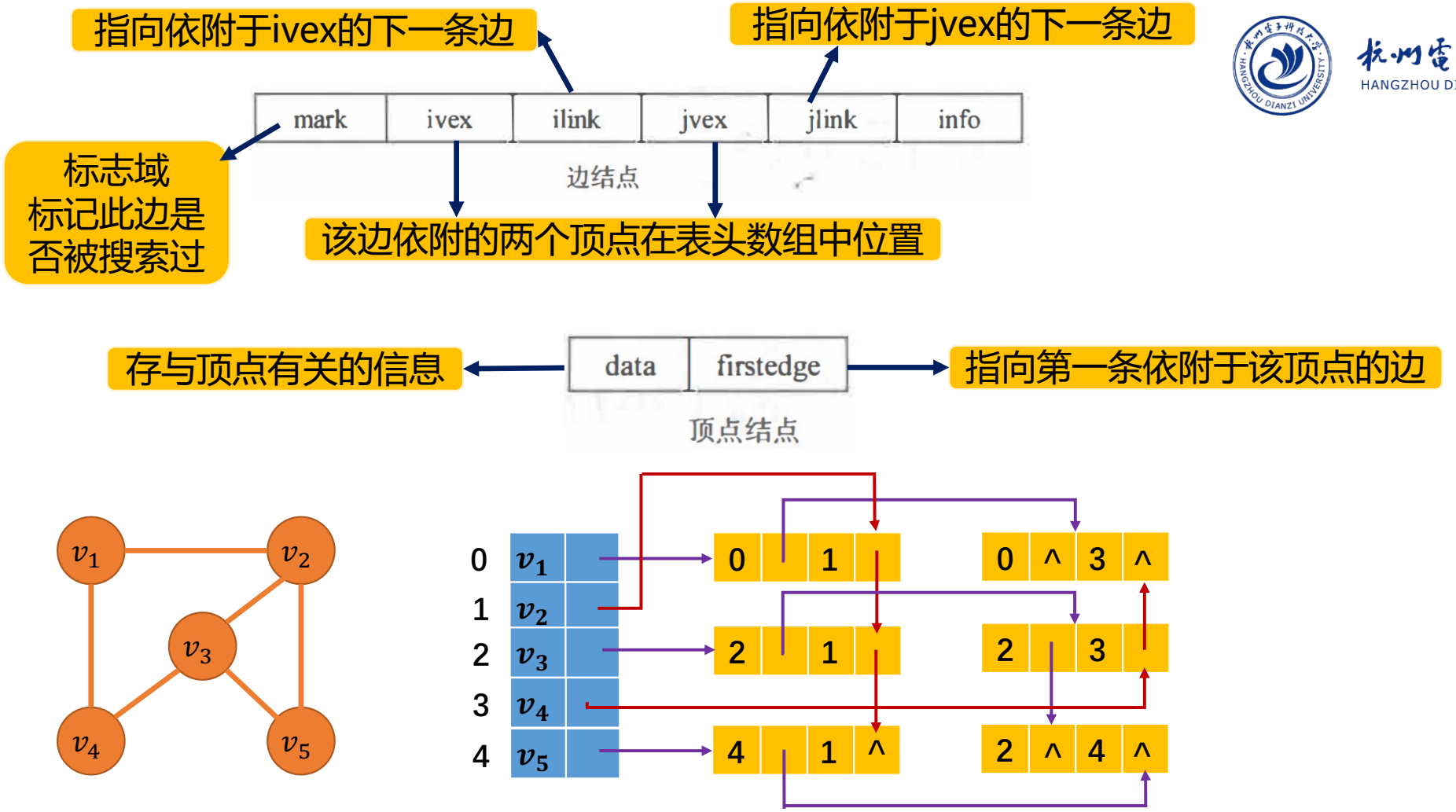
邻接表优点：容易求得顶点和边的信息。

缺点：某些操作不方便（如：删除一条边需找表示此边的两个结点）








指向依附于ivex的下一条边
指向依附于jvex的下一条边
mark
ivex
ilink
jvex
jlink
info
边结点
标志域
标记此边是
否被搜索过
该边依附的两个顶点在表头数组中位置
存与顶点有关的信息
data
firstedge
指向第一条依附于该顶点的边
顶点结点
v_1
v_2
v_3
v_4
v_5
0
1
2
3
4
^